spectrum$_{o}verloadDocumentation$

Release 0.3

Jason Neal

Apr 06, 2019

Contents

Spectrum_overload is to manipulate astronomical spectra in a spectrum class with *overloaded operators*. This means that you can easily divide and subtract spectra from each other using the math operators +, −, ⋆, /, ⋆⋆ keeping the wavelength, flux and headers together.

CHAPTER 1

Installation

Spectrum_overload is currently available from `github`.

Navigate to where you want to put files. Then Download:

with `git`:

```
$ git clone https://github.com/jason-neal/spectrum_overload.git
```

To install, run:

```
$ cd spectrum_overload
$ python setup.py install

# Or:

$ cd spectrum_overload
$ pip install .

# Or, for an editable installation:

$ cd spectrum_overload
$ pip install -e .
```

The plan is to have it available via `pypi` someday in the future.:

```
# Bug me about this.
$ pip install spectrum-overload
```

That day is not today…

If you have any issues with installation of spectrum_overload please open an issue so it can be resolved.

1.1 Requirements

The main requirements are

- numpy
- astropy
- scipy
- pyastronomy

If you are needing to use this package you probably have these installed already...

Unfortunately pyastronomy cannot be added to the `requirements.txt` alongside `numpy` due to issue #22 with the setup dependency of `numpy` in `pyastronomy`. If you do not have it installed already then run:

```
$ pip install pyastronomy
```

Other requirements are needed for running the pytest test suite. If you want to try run the tests, run:

```
$ python setup.py test
```

after normal installation.

These other dependencies that you may need are

- pytest-runner
- hypothesis
- pytest
- pytest-cov
- python-coveralls
- coverage

1.2 Editable Installation

If you want to modify spectrum_overload you can instead install it like:

```
$ python setup.py develop
```

CHAPTER 2

Quickstart Guide

First install spectrum_overload via the *installation guide*.

Import the Spectrum class into your code.

```
from spectrum_overload import Spectrum
```

Then to create a spectrum object, and raise to the power of 2 use:

```
s = Spectrum(flux=f, xaxis=x)

# power law scaling
s = s ** 2
```

where `f` and `x` are 1D lists or numpy arrays.

Note: Flux is the first kwarg. Be careful not to switch the order of flux and xaxis when not specifying the kwarg names.

By default the spectrum is "calibrated". If the xaxis is only the pixel values of your spectra you can pass:

```
s = Spectrum(flux=f, xaxis=pxls, calibrated=False)
```

or if no xaxis is given it will be set by

```
xaxis = np.arange(len(flux))
```

You can calibrate the spectra with a polynomial using the `calibrate_with` method which uses `np.polyval`:

```
s.calibrate_with([p0, p1, p2 ...])
```

Some methods are only available when you have a wavelength calibrated spectra, such as Doppler shifting with `Spectrum.doppler_shift()`.

2.1 Normalization

You can continuum normalize the spectrum using the `Spectrum.normalize()` method. It does not overwrite the spectrum but returns a new spectrum. Possible methods include `scalar`, `linear`, `quadratic`, `cubic`, `poly`, `exponential`. The `poly` method requires a `degree` to also be provided.

E.g.:

```
s = Spectrum(...)
s = s.normalize("linear")
# or
s = s.normalize("poly", degree=1)
```

The normalization happens by dividing th spectrum by the fitted continuum.

2.1.1 Continuum fitting

The fitting of the continuum is rather tricky due to the presence of absorption lines. The continuum is fitted by dividing the spectrum into N even bins. The highest M points out of each bin are chosen to represent the continuum for that bin. Their median/mean wavelength and flux position are used for each bin. A fit across the N bins is then performed using the specified method. The arguments `nbins` and `ntop` are used to specify the N and M values and can be passed to the `normalize` and `continuum` methods.

```
continuum = s.continuum(method"poly", degree=2, nbins=50, ntop=15)
```

You probably need to change these parameters to for the size/length of your spectrum.

Note: Its probably best to experiment with the best `nbins` and `ntop` parameters for your spectrum.

2.2 Overloaded Operators

The main purpose of this package is overloading the operators $+$, $-$, $\star$, $/$, $\star\star$ to work with spectrum objects. e.g:

```
# Given two spectra
s1 = Spectrum(...)
s2 = Spectrum(...)

# You can easily do the following operations.
add = s1 + s2
subtract = s1 - s2
multiply = s1 * s2
divide = s1 / s2
power = s1 ** a  # where a is a number
```

This is to make easier to do some basic manipulation of spectra, average spectra, take the difference, normalization, exponential scaling etc...

Note: Its probably best to interpolate the spectra to the same xaxis yourself before hand. If the spectra do not have the same wavelength axis then it is automatically spline interpolated to match the first spectrum or to another defined new xaxis.

CHAPTER 3

Available Classes

Currently there are two classes available.

- *Spectrum*
- *DifferentialSpectrum*

3.1 Spectrum

A class to contain a spectrum. The operators have been overloaded so that you can easily manipulate spectra.

class spectrum_overload.spectrum.**Spectrum**(**, xaxis: Union[numpy.ndarray, List[Union[int, float]], None] = None, flux: Union[numpy.ndarray, List[Union[int, float]], None] = None, calibrated: bool = True, header: Union[astropy.io.fits.header.Header, Dict[str, Any], None] = None, interp_method: str = 'spline'*)

Bases: object

Spectrum class to represent and manipulate astronomical spectra.

xaxis
The wavelength or pixel position values.

Type np.ndarray

flux
The extracted flux (measured intensity of light).

Type np.ndarray, array-like, list

calibrated
Flag to indicate calibration state. (Default = True.)

Type bool

header
Header information of observation.

Type astropy.Header, dict-like

add_noise(*snr: Union[float, int]*) → None
Add noise level of snr to the flux of the spectrum.

add_noise_sigma(*sigma*)
Add Gaussian noise with given sigma.

calibrate_with(*wl_map: Union[numpy.ndarray, List[int]]*) → None
Calibrate with a wavelength mapping polynomial.

Parameters **wl_map** – Polynomial coefficients of the form expected by np.poylval(). [p0, p1, p2 . . .]

Returns Replaces xaxis of self. Also self.calibrated is set to True.

Return type None

Notes

The parameters can be generated by np.polyfit(x, y, order)

continuum(*method: str = 'scalar', degree: Optional[int] = None, **kwargs*) → spectrum_overload.spectrum.Spectrum
Fit the continuum of the spectrum.

Fit a function of `method` to the median of the highest `ntop` points of `nbins` bins of the spectrum.‘‘

Parameters

- **method** (*str ("scalar")*) – The function type, valid functions are “scalar”, “linear”, “quadratic”, “cubic”, “poly”, and “exponential”. Default “scalar”.
- **degree** (*int, None*) – Degree of polynomial when method=”poly”. Default = None.
- **nbins** (*int*) – Number of bins to separate the spectrum into.
- **ntop** (*int*) – Number of highest points in bin to take median of.

Returns **s** – Spectrum of the continuum.

Return type *Spectrum*

copy() → spectrum_overload.spectrum.Spectrum
Copy the spectrum.

crosscorr_rv(*spectrum: spectrum_overload.spectrum.Spectrum, rvmin: float, rvmax: float, drv: float, **params*) → Tuple[numpy.ndarray, numpy.ndarray]
Perform pyasl.crosscorrRV with another spectrum.

Parameters

- **spectrum** (Spectrum) – Spectrum object to cross correlate with.
- **rvmin** (*float*) – Minimum radial velocity for which to calculate the cross-correlation function [km/s].
- **rvmax** (*float*) – Maximum radial velocity for which to calculate the cross-correlation function [km/s].
- **drv** (*float*) – The width of the radial-velocity steps to be applied in the calculation of the cross-correlation function [km/s].

- **params** (*dict*) – Cross-correlation parameters.

Returns

- **dRV** (*array*) – The RV axis of the cross-correlation function. The radial velocity refer to a shift of the template, i.e., positive values indicate that the template has been red-shifted and negative numbers indicate a blue-shift of the template. The numbers are given in km/s.
- **CC** (*array*) – The cross-correlation function.

Notes

The PyAstronomy function pyasl.crosscorrRV() is used

http://www.hs.uni-hamburg.de/DE/Ins/Per/Czesla/PyA/PyA/pyaslDoc/aslDoc/crosscorr.html

doppler_shift (*rv: float*) → None
Doppler shift wavelength by a given Radial Velocity.

Apply Doppler shift to the wavelength values of the spectrum using the radial velocity value provided and the relation RV/c = $\Delta\lambda/\lambda$.

Parameters **rv** (*float*) – Radial Velocity to Doppler shift by in km/s.

Warning:

Small RV : A lower limit of RV shift of 0.1 mm/s is set to prevent RV shifts much smaller than wavelength accuracy.

Uncalibrated xaxis : When the xaxis is uncalibrated there is no wavelength to Doppler shift. A message is printed and no shift is done.

Notes

The Doppler shift is calculated using the relation

$$RV/c = \Delta\lambda/\lambda$$

Where RV is the radial velocity (in km/s), λ_0 is the rest wavelength and $\Delta\lambda$ is the wavelength shift, $(\lambda_{shift} - \lambda_0)$

flux
Getter for the flux attribute.

instrument_broaden (*R*, ***pya_kwargs*)
Broaden spectrum by instrumental resolution R.

Uses the PyAstronomy instrBroadGaussFast function.

Parameters

- **R** (*int*) – Instrumental Resolution
- **pya_kwargs** (*dict*) – kwarg parameters for pyasl.instrBroadGaussFast()

Returns **s** – Broadened spectrum array.

Return type ndarray

interp_method
Getter for the interp_method attribute.

interpolate1d_to(*reference: Union[numpy.ndarray, str, Spectrum, List[int], List[float]], kind: str = 'linear', bounds_error: bool = False, fill_value: Union[str, numpy.ndarray] = nan*) → None

Interpolate wavelength solution to the reference wavelength.

This uses the scipy's interp1d interpolation. The optional parameters are passed to scipy's interp1d. This interpolates self to the given reference xaxis values. It overwrites the xaxis and flux of self with the new values.

Parameters

- **reference** (*Spectrum or numpy.ndarray*) – The reference xaxis values to interpolate to.
- **kind** (*(str or int, optional)*) – Specifies the kind of interpolation as a string ('linear', 'nearest', 'zero', 'slinear', 'quadratic, 'cubic' where 'slinear', 'quadratic' and 'cubic' refer to a spline interpolation of first, second or third order) or as an integer specifying the order of the spline interpolator to use. Default is 'linear'.
- **bounds_error** (*bool, optional*) – If True, a ValueError is raised any time interpolation is attempted on a value outside of the range of x (where extrapolation is necessary). If False, out of bounds values are assigned fill_value. By default, an error is raised unless fill_value="extrapolate".
- **fill_value** (*array-like or "extrapolate", optional (default = NaN)*) – If a ndarray (or float), this value will be used to fill in for requested points outside of the data range. If not provided, then the default is NaN. The array-like must broadcast properly to the dimensions of the non-interpolation axes. If a two-element tuple, then the first element is used as a fill value for x_new < x[0] and the second element is used for x_new > x[-1]. Anything that is not a 2-element tuple (e.g., list or ndarray, regardless of shape) is taken to be a single array-like argument meant to be used for both bounds as below, above = fill_value, fill_value. If "extrapolate", then points outside the data range will be extrapolated. ("nearest" and "linear" kinds only.)

Raises TypeError: – Cannot interpolate with the given object of type <type>.

References

scipy.interolate.interp1d https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html#scipy.interpolate.interp1d

length_check() → None

Check length of xaxis and flux are equal.

If everything is ok then there is no response/output.

Raises ValueError: – The length of xaxis and flux must be the same.

normalize(*method: str = 'scalar', degree: Optional[int] = None, **kwargs*) → spectrum_overload.spectrum.Spectrum

Normalize spectrum by dividing by the continuum.

Valid methods scalar, linear, quadratic, cubic, poly, exponential. poly method uses the degree value provided

Parameters

- **method** (*str ("scalar")*) – The function type, valid functions are "scalar", "linear", "quadratic", "cubic", "poly", and "exponential". Default "scalar".
- **degree** (*int, None*) – Degree of polynomial when method="poly". Default = None.

- **kwargs** – Extra parameters ntop and nbin for the `continuum` method.

Returns **s** – Normalized Spectrum.

Return type *Spectrum*

plot(*axis=None*, ***kwargs*) → None
Plot spectrum with matplotlib.

remove_nans() → spectrum_overload.spectrum.Spectrum
Returns new spectrum. Uses slicing with isnan mask.

shape()
Return flux shape.

spline_interpolate_to(*reference: Union[numpy.ndarray, str, Spectrum, List[int], List[float]], w: None = None, bbox: Optional[Any] = None, k: int = 3, ext: int = 0, check_finite: bool = True, bounds_error: bool = False*) → None

Interpolate wavelength solution using scipy's InterpolatedUnivariateSpline.

The optional parameters are for scipy's InterpolatedUnivariateSpline function.

Documentation copied from Sicpy:

One-dimensional interpolating spline for a given set of data points.

Fits a spline y = spl(x) of degree k to the provided x, y data. Spline function passes through all provided points. Equivalent to UnivariateSpline with s=0.

Parameters

- **x** (*(N,) array_like*) – Input dimension of data points this much be must be in ascending order.
- **y** (*(N,) array_like*) – Input dimension of data points
- **w** (*(N,) array_like, optional*) – Weights for spline fitting. Must be positive. If None (default), weights are all equal.
- **bbox** (*(2,) array_like, optional*) – 2-sequence specifying the boundary of the approximation interval. If None (default), bbox=[x[0], x[-1]].
- **k** (*int, optional*) – Degree of the smoothing spline. Must be 1 <= k <= 5.
- **ext** (*int or str, optional*) – Controls the extrapolation mode for elements not in the interval defined by the knot sequence.

  if ext=0 or 'extrapolate', return the extrapolated value. if ext=1 or 'zeros', return 0 if ext=2 or 'raise', raise a ValueError if ext=3 of 'const', return the boundary value.

  Default value is 0.
- **check_finite** (*bool, optional*) – Whether to check that the input arrays contain only finite numbers. Disabling may give a performance gain, but may result in problems (crashes, non-termination or non-sensical results) if the inputs do contain infinities or NaNs. Default is True.
- **Raises** –
- **-------** –
- **TypeError** – Cannot interpolate with the given object of type
- **ValueError** – A value in reference is outside the interpolation range."

See also:

https() //docs.scipy.org/doc/scipy-0.16.1/reference/generated/scipy.interpolate.InterpolatedUnivariateSpline.html#scipy.

wav_select(*wav_min: Union[float, int], wav_max: Union[float, int]*) → None
Select part of the spectrum between the given wavelength bounds.

Parameters

- **wav_min** (*float*) – Lower wavelength bound
- **wav_max** (*float*) – Upper wavelength bound

Returns Acts on self

Return type None

Notes

This might be better suited to return the new spectra instead of direct replacement.

xaxis
Getter for the xaxis attribute.

xlimits()

xmax()

xmin()

3.2 Differential Spectrum

Compute the difference between two *Spectrum* objects.

This is in an introductory state and need more work.

Would be useful add an s-profile function from Ferluga et al. 1997. The subtraction of two gaussian lines with a RV offset.

CHAPTER 4

Other Projects

There are many other packages that deal with spectra, none that I know of overload the operators.

In alphabetical order:

- astropy/specutils
- cokelaer/spectrum
- crawfordsm/specreduce
- pyspeckit/spectrum
- PySpectrograph/Spectra

I am sure there are others.

CHAPTER 5

Contributions

Any contributions and/or suggestions to improve this module are very welcome.

Submit issues, suggestions or pull requests to jason-neal/spectrum_overload.

This is my first attempt at creating classes, and packaging a python project so any *helpful* feedback is appreciated.

CHAPTER 6

Badges

master

develop

Note: To build the documentation locally go to the root *spectrum_overload* directory then run:

```
pip install -e .[docs]   # editable installation, install docs dependencies.
cd docs
make html
```

The documentation pages will be docs/.build/index.html.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

X